R_ForLoops.R

Administrator

Thu Feb 11 14:52:29 2016

```
# Understanding For loops
# 11 February 2016
# NJG

# for (var in seq) { }

# var changes in the loop

for (i in 1:5) {
  cat("i =",i,"\n")
}
```

```
## i = 1
## i = 2
## i = 3
## i = 4
## i = 5
```

```
# better to set a variable as the endpoint

z <- 5
for (i in 1:z) {
  cat("i=",i," z=",z,"\n")
}
```

```
## i= 1  z= 5
## i= 2  z= 5
## i= 3  z= 5
## i= 4  z= 5
## i= 5  z= 5
```

```
# be careful about the sequence endpoints

for (i in z) {
  cat("i=",i," z=",z,"\n")
}
```

```
## i= 5  z= 5
```

```
# be careful about variables inside and outside of the loop and how they are initialized

z <- 5
v1 <- 10
v2 <- 0
for (i in 1:z) {
```

```r
  v1 <- v1 + i^2
  v2 <- v2*i
  cat("i= ",i," z=",z," v1=",v1," v2=",v2,"\n")
}
```

```
## i=  1  z= 5  v1= 11  v2= 0
## i=  2  z= 5  v1= 15  v2= 0
## i=  3  z= 5  v1= 24  v2= 0
## i=  4  z= 5  v1= 40  v2= 0
## i=  5  z= 5  v1= 65  v2= 0
```

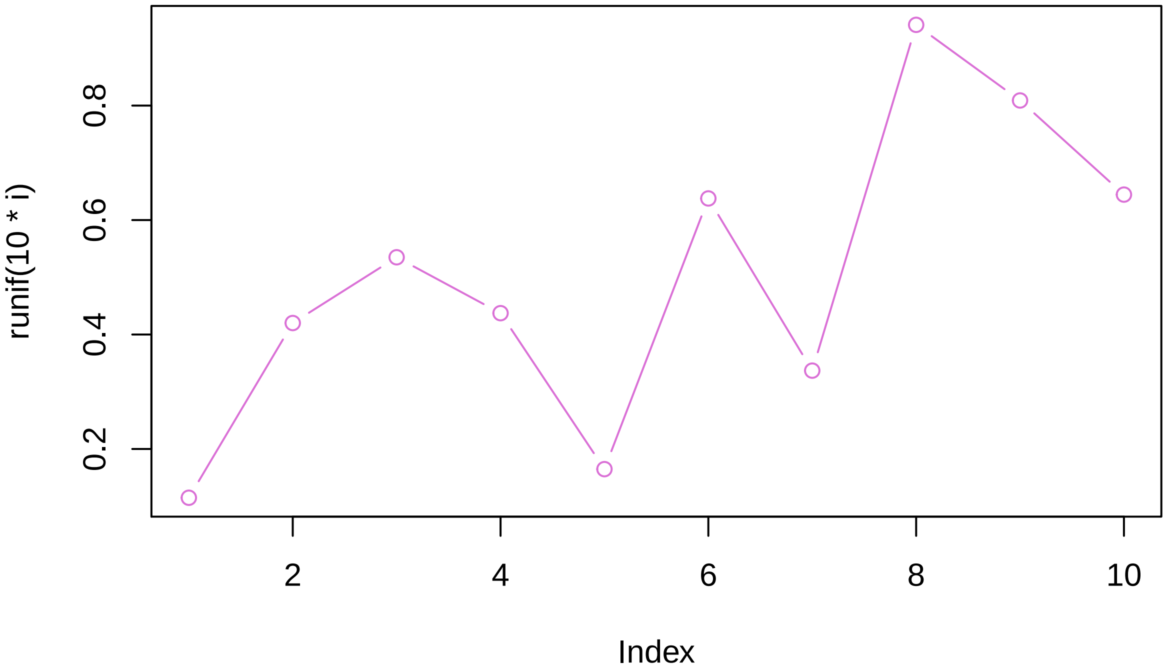
```r
# always make the loop variable an integer sequence

# permissible, but very confusing

z <- c(1.1, 4.4, 5.5)
for (i in z) {
  result <- i^2
  cat("i = ",i, "result = ",result," z = ",z,"\n")
}
```

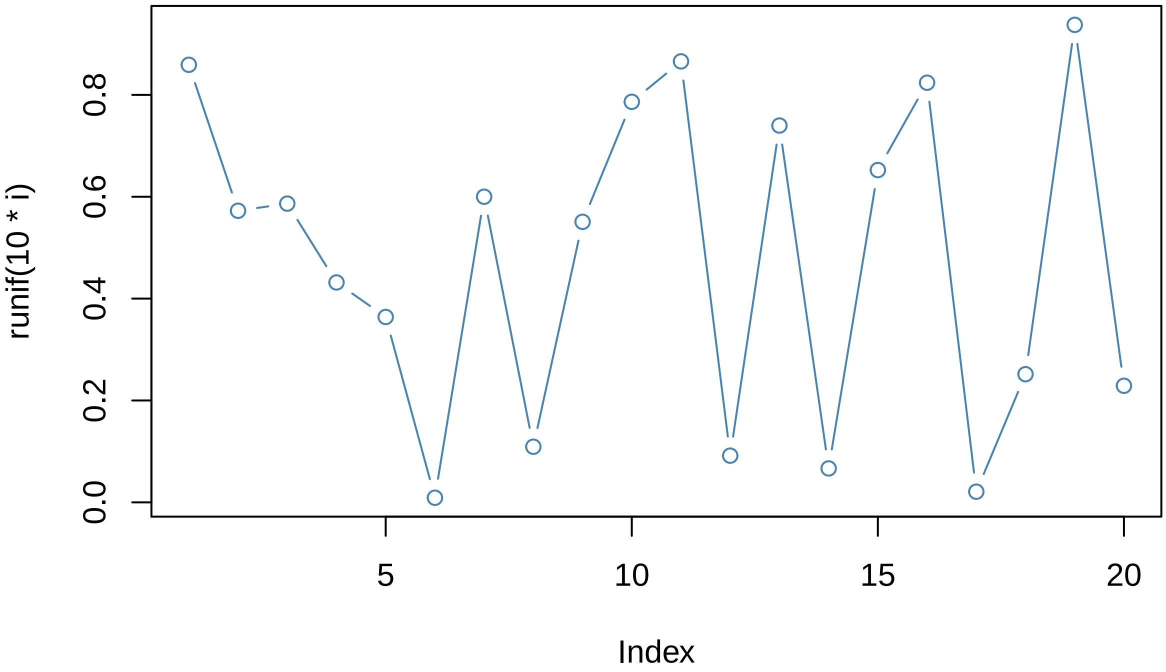
```
## i =  1.1 result =  1.21  z =  1.1 4.4 5.5
## i =  4.4 result =  19.36  z =  1.1 4.4 5.5
## i =  5.5 result =  30.25  z =  1.1 4.4 5.5
```

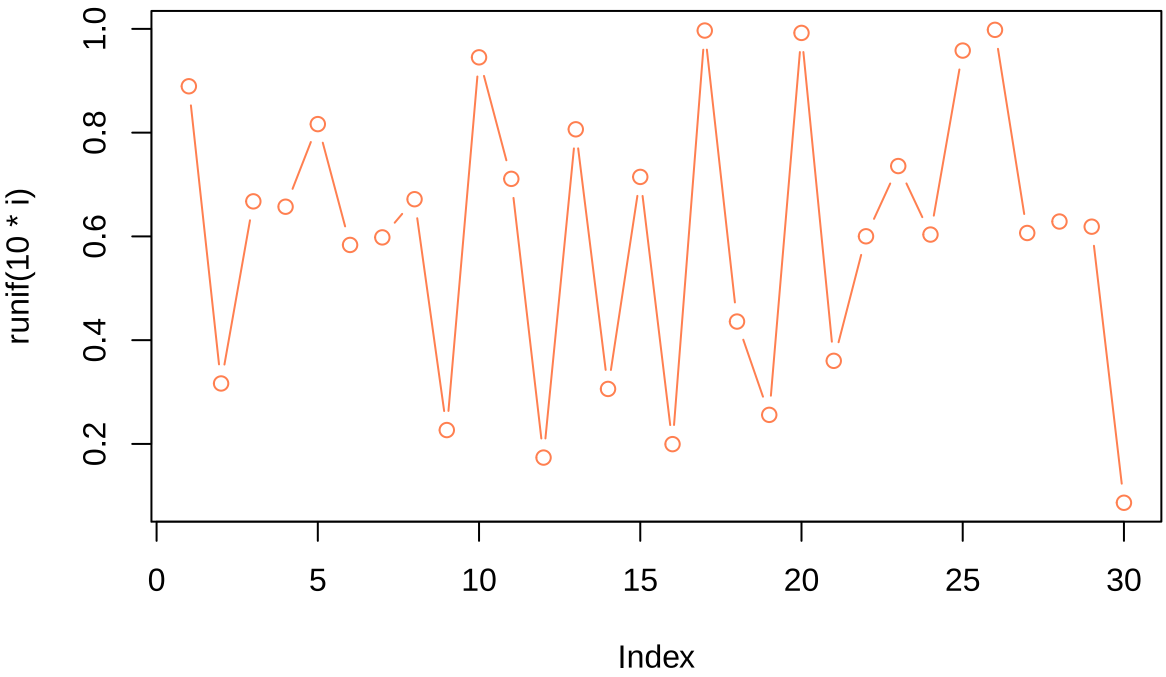
```r
myvec <- c(1.1, 4.4, 5.5)

for (i in 1:length(myvec)) {
  result <- myvec[i]^2
  cat("i = ",i," result = ",result," length(myvec) = ",length(myvec),"\n")
}
```

```
## i =  1  result =  1.21  length(myvec) =  3
## i =  2  result =  19.36  length(myvec) =  3
## i =  3  result =  30.25  length(myvec) =  3
```

```r
# still easy to operate on character strings

mycolors <- c("orchid","steelblue","coral")

for (i in 1:length(mycolors)) {
  plot(runif(10*i),type="b",col=mycolors[i])
}
```


runif(10 * i)
0.2
0.4
0.6
0.8
2
4
6
8
10
Index

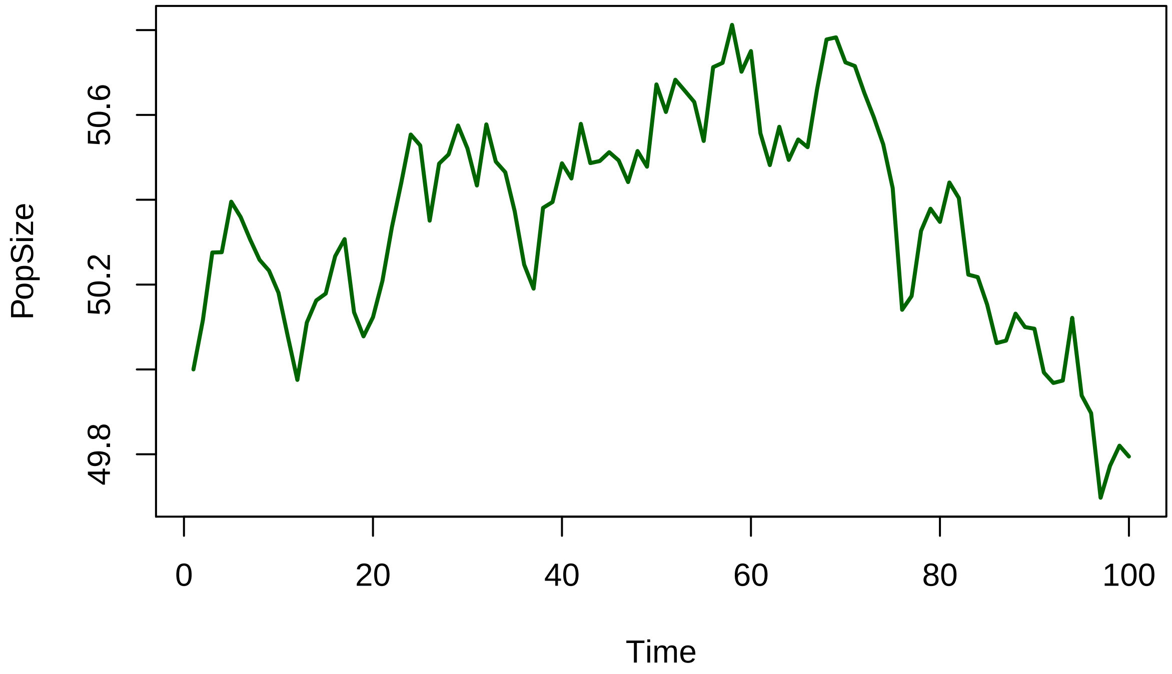

runif(10 * i)
0.0
0.2
0.4
0.6
0.8
5
10
15
20
Index



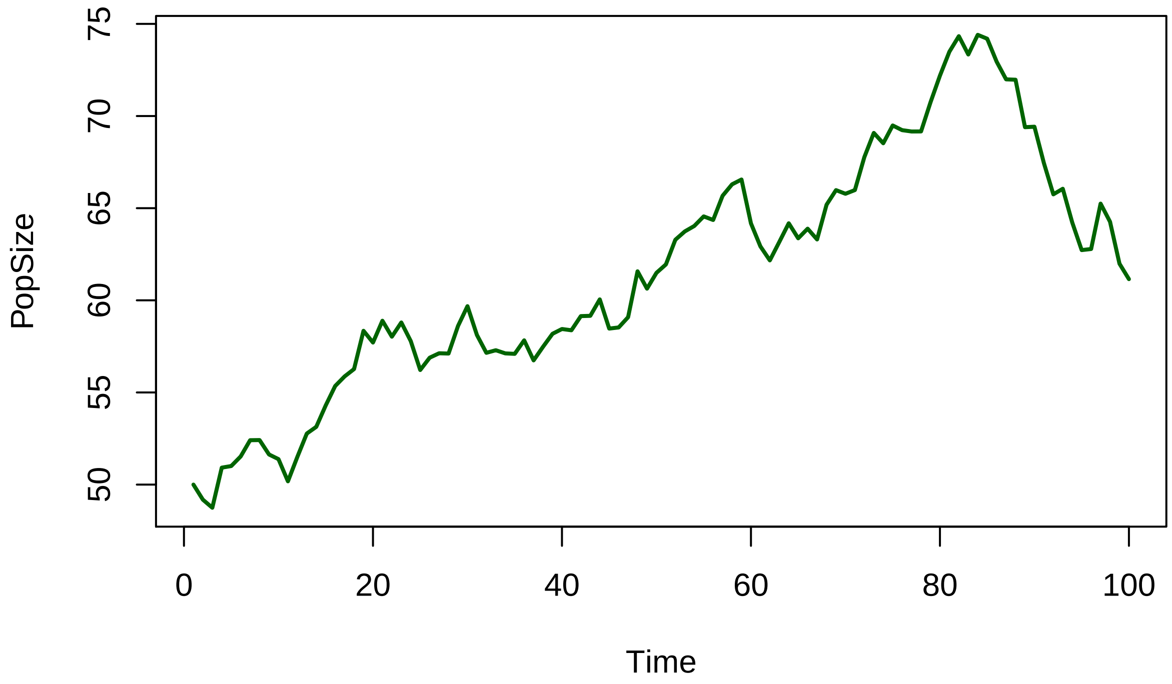
```
# A random walk model of population growth!
# Referencing different elements in the vector

# Define 2 global variables
N0 <- 50 #  initial population size
time <- 100 # length of time series

PopSize <- vector(mode="numeric",length=time)
head(PopSize)
```

```
## [1] 0 0 0 0 0 0
```

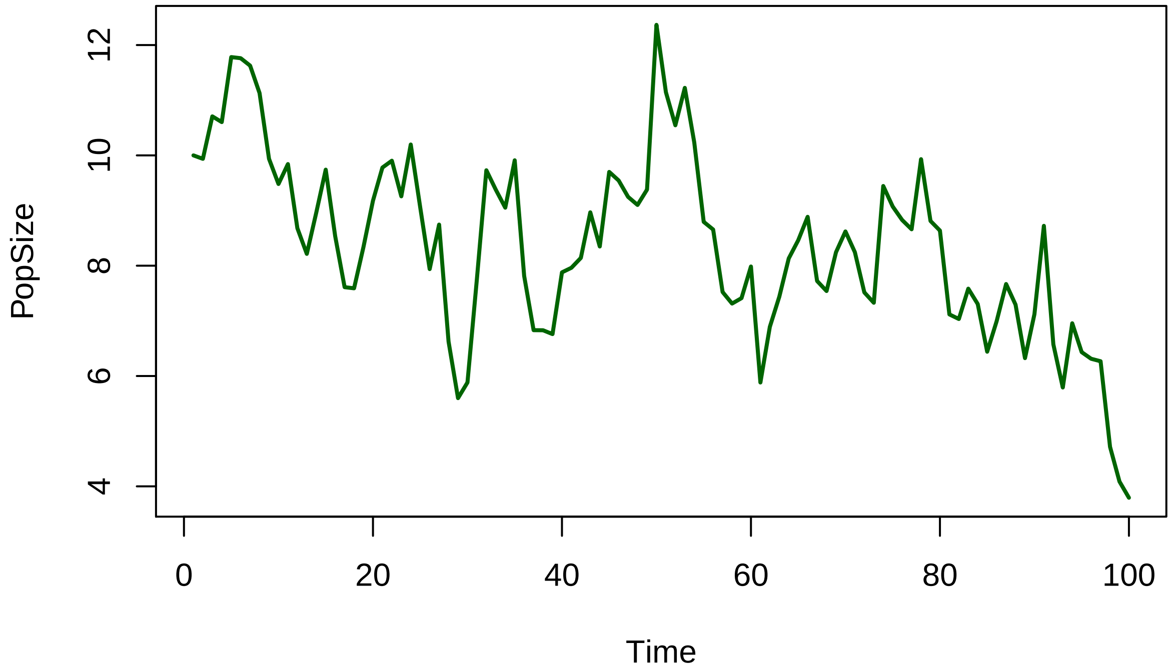
```
PopSize[1] <- N0

for (i in 2:length(PopSize)) {
  PopSize[i] <- PopSize[i - 1] + rnorm(n=1, mean= 0.0, sd=0.1)
}

plot(x=1:length(PopSize),y=PopSize,type="l",xlab="Time",ylab="PopSize",col="darkgreen",lwd=2)
```



```
# Bundle In A Function
RandomWalk <- function(N0=50,time=100,mean=0,sd=1){
PopSize <- vector(mode="numeric",length=time)
PopSize[1] <- N0

for (i in 2:length(PopSize)) {
  PopSize[i] <- PopSize[i - 1] + rnorm(n=1, mean= mean, sd=sd)
}

plot(x=1:length(PopSize),y=PopSize,type="l",xlab="Time",ylab="PopSize",col="darkgreen",lwd=2)
}

RandomWalk()
```



```
# Use the break function to stop a loop before the end

RandomWalk2 <- function(N0=50,time=100,mean=0,sd=1){
PopSize <- vector(mode="numeric",length=time)
PopSize[1] <- N0

for (i in 2:length(PopSize)) {
  PopSize[i] <- PopSize[i - 1] + rnorm(n=1, mean= mean, sd=sd)
  if (PopSize[i] <= 0) {
    PopSize[i] <- 0
    break
  }
}

plot(x=1:length(PopSize),y=PopSize,type="l",xlab="Time",ylab="PopSize",col="darkgreen",lwd=2)
}
RandomWalk2(N0=10,time=100)
```



```
# using a double for loop to traverse a matrix

m <- matrix(1:12,nrow=4,byrow=TRUE)

for (i in 1:nrow(m)) {
  for (j in 1:ncol(m)) {
    z <- runif(1)
    m[i,j] <- m[i,j] + z
  }
}
print(m)
```

```
##           [,1]      [,2]      [,3]
## [1,]  1.757148  2.429732  3.635259
## [2,]  4.646364  5.948370  6.325738
## [3,]  7.435808  8.878563  9.164080
## [4,] 10.112403 11.252142 12.627553
```

```
# move the random number outside the inner loop

m <- matrix(1:12,nrow=4,byrow=TRUE)

for (i in 1:nrow(m)) {
    z <- runif(1)
  for (j in 1:ncol(m)) {
    m[i,j] <- m[i,j] + z
  }
}
print(m)
```

```
##           [,1]      [,2]      [,3]
## [1,]  1.735770  2.735770  3.735770
## [2,]  4.309758  5.309758  6.309758
## [3,]  7.998907  8.998907  9.998907
## [4,] 10.052623 11.052623 12.052623
```

```
# move the random number outside entire loop

m <- matrix(1:12,nrow=4,byrow=TRUE)
    z <- runif(1)

for (i in 1:nrow(m)) {
  for (j in 1:ncol(m)) {
    m[i,j] <- m[i,j] + z
  }
}
print(m)
```

```
##           [,1]      [,2]      [,3]
## [1,]  1.822648  2.822648  3.822648
## [2,]  4.822648  5.822648  6.822648
## [3,]  7.822648  8.822648  9.822648
## [4,] 10.822648 11.822648 12.822648
```

```
# Q: How to fill by columns?
# A: reverse row and column loops
m <- matrix(1:12,nrow=4,byrow=TRUE)

for (i in 1:ncol(m)) {
    z <- runif(1)
  for (j in 1:nrow(m)) {
    m[j,i] <- m[j,i] + z
  }
}
print(m)
```

```
##           [,1]      [,2]      [,3]
## [1,]  1.136705  2.598981  3.145138
## [2,]  4.136705  5.598981  6.145138
## [3,]  7.136705  8.598981  9.145138
## [4,] 10.136705 11.598981 12.145138
```

```
# Acting on matrices without for loops

# Add a different random number to each element
m <- matrix(1:12,nrow=4,byrow=TRUE)
m <- m + runif(n=length(m))
print(m)
```

```
##           [,1]      [,2]      [,3]
## [1,]  1.864543  2.996760  3.604810
## [2,]  4.384904  5.782621  6.558888
## [3,]  7.617942  8.639488  9.174016
## [4,] 10.835517 11.285256 12.080047
```

```
# Add the same random number to all elements
m <- matrix(1:12,nrow=4,byrow=TRUE)
m <- m + runif(n=1)
print(m)
```

```
##          [,1]     [,2]     [,3]
## [1,]  1.69503  2.69503  3.69503
## [2,]  4.69503  5.69503  6.69503
## [3,]  7.69503  8.69503  9.69503
## [4,] 10.69503 11.69503 12.69503
```

```
# Combine for loops and vectorized operations
# Add the same number to each row
m <- matrix(1:12,nrow=4,byrow=TRUE)
for (i in 1:nrow(m)) {
  m[i,] <- m[i,] + runif(1)
}
print(m)
```

```
##           [,1]      [,2]      [,3]
## [1,]  1.427626  2.427626  3.427626
## [2,]  4.416504  5.416504  6.416504
## [3,]  7.452085  8.452085  9.452085
## [4,] 10.121102 11.121102 12.121102
```

```
# Add the same number to each column
m <- matrix(1:12,nrow=4,byrow=TRUE)
for (j in 1:ncol(m)) {
  m[,j] <- m[,j] + runif(1)
}
print(m)
```

```
##           [,1]      [,2]      [,3]
## [1,]  1.135544  2.968763  3.651427
## [2,]  4.135544  5.968763  6.651427
## [3,]  7.135544  8.968763  9.651427
## [4,] 10.135544 11.968763 12.651427
```